

On complexity of regular realizability problems^{*}

Mikhail N. Vyalyi
vyalyi@gmail.com

December 29, 2012

A regular realizability (RR) problem is testing nonemptiness of intersection of some fixed language (filter) with a regular language. We show that RR problems are universal in the following sense. For any language L there exists RR problem equivalent to L under disjunctive reductions on nondeterministic log space.

We conclude from this result an existence of complete problems under polynomial reductions for many complexity classes including all classes of the polynomial hierarchy.

Motivation of this work is to find out a specific class of algorithmic problems that represents in a unified way complexities of all known complexity classes (there are hundreds of them now).

A typical algorithmic problem is recognition problem for a language. But in the most interesting cases an input is structured: it is a graph, function description etc. Our main goal is to choose a structure of an input to satisfy two (somewhat contradictory) requirements: a specific class of problems should be wide enough and it should be useful. The latter requirement reflects a hope that analysis of a specific problem might be easier than a general case.

Here we consider regular realizability problems in this context. Let L be a language (we refer it further as *a filter*). The *regular realizability* problem with this filter is a question about realizability of regular properties on words in L . More exactly, the language $RR(L)$ consists of descriptions of regular languages R such that $R \cap L \neq \emptyset$.

What are possible complexities of RR problems? In this paper we obtain a partial answer on this question. It appears that RR problems are universal: for any other problem there exists an equivalent RR problem.

To make exact statements we need to fix a format for descriptions of regular languages and an equivalence relation.

We represent a regular language R by a deterministic finite automaton (DFA) A recognizing the language R (and denote this fact as $R = L(A)$). DFAs are described in natural way by transition tables. Details of the format used can be found in [1]. Important features for this work are: (i) each binary word w is a description of some DFA $A(w)$ and (ii) testing membership for a regular language $L(w) = L(A(w))$ can

^{*}Supported by RFBR grants 11-01-00398 and 12-01-00864

be done using deterministic log space. So formal definition of the language $RR(L)$ corresponding to RR problem with a filter L is

$$w \in RR(L) \Leftrightarrow L \cap L(A(w)) \neq \emptyset.$$

Equivalence relations considered here are induced by algorithmic reductions. For any language X an RR-representative of X (under reductions of some type) is a language L such that $RR(L)$ is equivalent to X under reductions of this type.

The most natural reductions in the context of regular realizability are m -reductions using log space (\leq_m^{\log} -reductions). There exist RR problems complete under log space reductions for complexity classes such as LOG, NLOG, P, NP, PSPACE, EXP, Σ_1 , see [2, 3].

We can prove universality result mentioned above for stronger reductions: disjunctive reductions using nondeterministic log space (definition see below in Section 1). In the proof we reduce an arbitrary language X to RR language by a *monoreduction*: a reduction of special kind that sends a word w to a description of DFA accepting 1-element set $\{f(w)\}$, where f is an injective map. Disjunctive nlog space reductions appear in an attempt to invert a monoreduction (see Section 4).

For many cases disjunctive nlog space reductions are weaker than polynomial reductions. In this way we extend a list of classes having complete RR problems under polynomial reductions. It will be shown in Section 2 that there exist RR complete problems for each class of the polynomial hierarchy. Note that it is a rather surprising even for the class co-NP: RR problem is formulated by existential quantifier (an existence of an accepting path possessing specific properties) and there is no direct way to convert it into universal quantifier.

We also present two other universality results.

In Section 3 we show that RR problems are universal for promise problems. In this case inversion of a monoreduction is much simpler. But it requires FNL reductions too. It seems that using nondeterministic log space is unavoidable (see Remark 3 in Section 3).

In Section 5 we extend universality to the classes of generalized nondeterminism introduced in [3]. Technically it suffices to prove universality of RR problems with prefix closed filters. The proofs in this section are similar to the proofs in Section 4 but they involve more technical details.

Thus RR problems are universal. They represent a huge variety of complexities. Are they useful? The proof suggests a negative answer. Reductions used in the proof cut off almost all properties of regular languages and put ‘the hard part’ of a problem into instances corresponding to finite languages. Of course, nlog reductions say nothing about languages inside NLOG.

1 What reductions are used in universality results

Equivalence relations used here are defined using reductions. Languages are equivalent (notation $A \sim B$) if they are reduced to each other under reductions of some type.

We recall some basic definitions concerning algorithmic reductions. Let \mathcal{C} be a function class. A language A is reduced to a language B under m -reductions by func-

tions from the class (notation \leq_m^C) if there exists a function $f \in C$ such that $x \in A$ iff $f(x) \in B$. If the class C contains the identity map and is closed under compositions then m -reduction relation is a preorder, i.e. transitive and reflexive relation.

The most known reductions of this type are polynomial reductions (see, e.g. [4]).

In this paper we need weaker m -reductions. The corresponding functions are computed by deterministic Turing machines using space logarithmically bounded w.r.t. the input length. We denote these reductions by \leq_m^{\log} . More exactly, the machine has the read only input tape, the work tape of size $O(\log n)$, where n is an input size and write only output tape. It easy to check that the relation \leq_m^{\log} is transitive. Note that the size of $f(x)$ is polynomially bounded by the size of x (the number of configurations is polynomially bounded). So, the log space algorithm for a composition $f(g(x))$ uses a subroutine to compute i th bit of $g(x)$ to simulate the input $g(x)$ for the algorithm computing f . More details can be found in textbooks on complexity theory, say, [4, 5].

The second important type of reductions is Turing reductions. A language A is Turing reducible to a language B if there exists an algorithm recognizing A that uses an oracle B . There are several restricted forms of Turing reductions. In *truth-table reductions* a reducing algorithm generates a list of oracle queries q_1, \dots, q_s and a Boolean function α depending on s arguments. Then algorithm asks all queries and outputs $\alpha(\chi_B(q_1), \dots, \chi_B(q_s))$.

Note that for log space Turing reductions and truth-table reductions are equivalent [6].

We use in the main result a weaker form of truth-table reductions—disjunctive reductions (notation \leq_{dtt}). In this case the function α is disjunction.

Disjunctive reductions can be expressed via m -reductions in the following way. Define a language $\text{Seq}(X)$ as a collection of words in the form

$$\#x_1\#x_2\#\dots x_n\#,$$

where $x_i \in X$ for some i , $\#$ is a delimiter (an additional symbol that do not belong to the alphabet of the language X). The following statement is clear from definition.

Proposition 1. $A \leq_{\text{dtt}} B$ iff $A \leq_m \text{Seq}(B)$.

Words from $\text{Seq}(X)$ can be identified with finite sequences of words from X in natural way. Hereinafter we assume this correspondence.

Definition 1. A class of languages is *normal* if it is closed under the map $X \mapsto \text{Seq}(X)$ and \leq_m^{\log} -reductions: if $X \in C$ and $Y \leq_m^{\log} X$ then $\text{Seq}(X) \in C$ and $Y \in C$.

Definition 2. A language X is \leq_m -*normal* if $X \sim \text{Seq}(X)$ under \leq_m -reductions.

If a function class is not indicated we assume log space reductions.

Proposition 2. $\text{Seq}(X)$ is normal for any X .

Proof. It is obvious that $Y \leq_m^{\log} \text{Seq}(Y)$ for any Y . Thus we need to prove that $\text{Seq}(\text{Seq}(X)) \leq_m^{\log} \text{Seq}(X)$. Note that

$$\text{Seq}(\text{Seq}(X)) \leq_{\text{dtt}}^{\log} X.$$

Indeed, a disjunction of disjunctions is a disjunction. So the reducing algorithm form a query list consisting of all elements of all elements of an input sequence.

To complete a proof apply Proposition 1. \square

Lemma 1. *Let $\text{Seq}(X) \leq_m^{\log} Y$, $Y \leq_m^{\log} \text{Seq}(X)$. Then Y is normal.*

Proof. We denote reducing maps by $f: (x_1, \dots, x_t) \mapsto y$ (for the first reduction) and $g: y \mapsto (x_1, \dots, x_t)$ (for the second one).

Now we reduce $\text{Seq}(Y)$ to Y . Compute sequences $g(y_i)$ for all elements y_i of the sequence (y_1, \dots, y_s) and merge all of them into a sequence $h(y)$ consisting of words from the language X . Apply to this sequence the map f . It is a reduction in question. Correctness stems from the fact that $y \in \text{Seq}(Y)$ iff $h(y) \in \text{Seq}(X)$ iff $f(h(y)) \in Y$. \square

Corollary 1. *If a normal class contains m -complete languages then all complete languages in this class are normal.*

Now we discuss a choice of a function class for disjunctive reductions.

It is natural that we prefer the weakest possible class. Log space seems to be insufficient (see Remark 3 below).

Next step is to use nondeterministic space. The corresponding class is denoted by FNL. Machines computing FNL functions use a logarithmically bounded work tape and an unbounded oracle tape which is one way and write only. An oracle puts its answer on the oracle tape and overwrites a query. More details on the class FNL and its companions can be found in [7]. In particular, the composition lemma holds for the class FNL and the size of output is polynomially bounded by the input size.

We denote FNL disjunctive reductions by $\leq_{\text{dtt}}^{\text{FNL}}$. The FNL m -reduction is denoted by \leq_m^{FNL} .

It is clear from definition that $\leq_{\text{dtt}}^{\text{FNL}}$ -reductions are stronger than log space reductions and are weaker than disjunctive reductions in polynomial time. So, we have the following facts.

Proposition 3. *If $A \leq_{\text{dtt}}^{\log} B$ then $A \leq_{\text{dtt}}^{\text{FNL}} B$.*

Proposition 4. *If $A \leq_{\text{dtt}}^{\text{FNL}} B$ then $A \leq_{\text{dtt}}^{\text{P}} B$. Moreover, if B is normal then $A \leq_m^{\text{FNL}} B$.*

Remark 1. The universality is much easier to prove for exponential time reductions. But these reductions do not say anything about the most interesting realm of complexities— PSPACE and below.

For polynomial time reductions we need the same constructions that are used in our proof below. The algorithmic facts become easier.

We use FNL reductions to cover P and below. Note that as shown in [8] basic counting log space classes are closed under $\leq_{\text{dtt}}^{\text{FNL}}$ -reductions.

2 Examples of normal classes

We will apply Corollary 1 and $\leq_{\text{dtt}}^{\text{FNL}}$ -universality results to prove that a complexity class contains complete RR problems. Corollary 1 requires a class into consideration to be normal.

The most of known complexity classes are normal. We give several examples. In definitions of complexity classes and computational models we follow Arora and Barak book [4].

A straightforward algorithm for recognizing language $\text{Seq}(X)$ is to check $x_i \in X$ for all x_i from the input

$$\#x_1\#\dots\#x_m\#$$

and take disjunction of the results.

Let $X \in \text{DSPACE}(f(n))$, where $f(n) = \Omega(\log n)$. Then the above algorithm uses space $O(f(n))$. Thus the class $\text{DSPACE}(f(n))$ is normal (it closed under \leq_m^{\log} -reductions by obvious reasons).

With small modifications the same argument is applied to nondeterministic space classes. Nondeterministic algorithm guesses¹⁾ i such that $x_i \in X$ and runs an algorithm recognizing X on an instance x_i . So the classes $\text{NSPACE}(f(n))$, $f(n) = \Omega(\log n)$, are also normal.

For time complexity classes closeness under \leq_m^{\log} -reductions holds for the class P of polynomial time and more powerful classes. Running time of the above algorithm recognizing $\text{Seq}(X)$ is upperbounded by

$$\tilde{T}(n) = n + \max_{n=n_1+\dots+n_m} (T(n_1) + \dots + T(n_m)), \quad (1)$$

where $T(n)$ is a running time of an algorithm recognizing X . It is clear that $T(n) = \text{poly}(n)$ implies $\tilde{T}(n) = \text{poly}(n)$. So P is normal. For more powerful classes normality of a time complexity class follows from closeness of time limitations under the map $T(n) \mapsto \tilde{T}(n)$. There is a simple sufficient condition for closeness: if a function $T(n)$ satisfies time limitations then $nT(n)$ is also satisfies time limitations. Applying this observation we get normality for classes of quasipolynomial time, exponential time, simple exponential time and similar limitations.

It is easy to see that under the same conditions nondeterministic time classes are also normal.

The last example are classes of the polynomial hierarchy.

Proposition 5. *Each class of the polynomial hierarchy is normal.*

Proof. Closeness under \leq_m^{\log} -reductions is clear for each class of polynomial hierarchy. So it remains to show closeness under the map $X \mapsto \text{Seq}(X)$.

Let $\text{ISeq}(X)$ be a language consisting of $\#\langle i \rangle \#x_1 \dots \#x_m \#$ such that $x_i \in X$. Here the delimiter symbol $\#$ does not belong to the alphabet of the language X and $\langle i \rangle$ denotes binary representation of integer i . It is clear that $X \sim_m^{\log} \text{ISeq}(X)$. Indeed, using log space one can extract i th list element.

By definition of $\text{ISeq}(X)$ we have

$$(w \in \text{Seq}(X)) \Leftrightarrow \exists i (\#iw \in \text{ISeq}(X)). \quad (2)$$

Note that if $X \in \Sigma_k^P$ then $\{\#iw : \#iw \in \text{ISeq}(X)\}$ is in Σ_k^P . Thus $\text{Seq}(X)$ is in Σ_k^P due to (2).

¹⁾Hereinafter a guess is a nondeterministic choice.

Suppose now that $X \in \Pi_k^p$. In this case $\{\#iw : \#iw \in \text{ISeq}(X)\}$ is in Π_k^p and we have for some $V \in \Sigma_{k-1}^p$ and polynomial $p(\cdot)$

$$(\#iw \in \text{ISeq}(X)) \Leftrightarrow \forall y(|y| \leq p(|w|)) \wedge (\#iw\#y \in V).$$

So we need to interchange quantifiers in (2). Due to this fact one can apply an equality

$$\exists i \in S \forall y W(x, i, y) = \forall_{i \in S} y^i \bigvee_{i \in S} W(x, i, y^i),$$

where y^i are new variables indexed by $i \in S$. Thus $\text{Seq}(X) \in \Pi_k^p$. \square

3 Monoreductions

Without loss of generality we consider languages in binary alphabet.

Let f be an injective map $\{0, 1\}^* \rightarrow \{0, 1\}^*$.

A *monoreduction* is a map

$$x \mapsto A_{f(x)}, \quad (3)$$

where A_w is a description of minimal DFA²⁾ recognizing the 1-element language $\{w\}$. Note that the number of states in minimal DFA coincides with the number of Myhill – Nerode classes [11]. It is easy to verify that the number of Myhill – Nerode classes for the language $\{w\}$ is just $|w| + 2$ (all prefixes of the word w plus one) and the function $w \mapsto A_w$ is computed on deterministic log space.

Informally speaking, the map (3) assigns ‘names’ for all binary words in the form of automaton description. Injectivity condition implies that names of different words are different.

If a map $f(x)$ is log space computed then the corresponding monoreduction is also log space computed. It reduces a nonempty language $\emptyset \subset X \subseteq \{0, 1\}^*$ to some RR problem

$$X \leq_m^{\log} \text{RR}(Y). \quad (4)$$

It is easy to see from definitions that (4) holds iff

$$Y \cap f(\bar{X}) = \emptyset, \quad Y \supseteq f(X). \quad (5)$$

In other words, Y separates images of X and \bar{X} and Y contains the image of X .

Complexity of a reduction in the opposite direction depends heavily on Y .

Take for example $Y = f(X)$. It is not a good choice because complexity of the language $\text{RR}(f(X))$ varies in wide range w.r.t. complexity of the language X . It can be illustrated in the simplest case $f = \text{id}$.

There exists a filter L such that (a) the membership problem for L is in the class of languages recognized by RAM in linear time; (b) $\text{RR}(L)$ is complete for the class Σ_1 of recursively enumerable languages under m -reductions [3].

²⁾We assume that construction of minimal automaton is fixed. For minimization algorithms see textbooks on formal languages, e.g. [11].

On the other hand for a $X = \{0^n\}$ a language $\text{RR}(X)$ is in LOG. It was shown in [1] that LOG is \leq_m^{\log} -reducible to any RR problem with infinite filter.

Nevertheless, the reduction $X \leq_m^{\log} \text{RR}(X)$ can be inverted if we consider reductions among promise problems. A *promise problem* is a problem of computing a partially defined predicate. In other words there are two languages L_1 and L_0 such that $L_1 \cap L_0 = \emptyset$. The question is to test membership $w \in L_1$ provided either $w \in L_1$ or $w \in L_0$.

Promise problems have more expressive power than languages (which correspond to total predicates). For many complexity classes, say $\text{NP} \cap \text{co-NP}$ or BPP, an existence of complete languages in a class is an open problem. But there are simple and natural examples of complete promise problems for these classes.

The question about complete RR promise problems is also much easier than the question about complete RR languages.

Theorem 1. *Any promise problem (L_1, L_0) with $L_1 \neq \emptyset$ is equivalent to RR promise problem under $n \log$ space reductions.*

Proof. Define $\text{RR}(L_1 : |R| = 1)$ as RR problem with a promise $|L(A)| = 1$, i.e. a pair of languages

$$(\{A : L(A) \cap L_1 \neq \emptyset \wedge |L(A)| = 1\}, \{A : L(A) \cap L_1 = \emptyset \wedge |L(A)| = 1\}).$$

Then a promise problem (L_1, L_0) is \leq_m^{\log} -reduced to the problem $\text{RR}(L_1 : |R| = 1)$ by the map $w \mapsto A_w$.

In the other direction we can prove a weaker reduction

$$\text{RR}(L_1 : |R| = 1) \leq_m^{\text{FNL}} (L_1, L_0). \quad (6)$$

To construct a reduction (6) we need a procedure that find a word accepted by DFA A provided A recognizes a 1-element language. This procedure is easily implemented in the class FNL: it nondeterministically guesses the word symbol by symbol maintaining the current state of the automaton reading the word. \square

Remark 2. We call an automaton *simple* if it recognizes a 1-element set. Testing simplicity can be done using $n \log$ space.

Testing algorithm again guesses a word accepted by the automaton symbol by symbol. But now it also checks a possibility to choose more than one symbol in such a way that after each choice the automaton can be moved to an accepting state by reading some sequence of symbols. Here we use a well-known equality $\text{co-NLOG} = \text{NLOG}$ [9, 5].

Remark 3. Is it necessary to use NLOG-oracle in the reduction (6)? The question is open but the negative answer is more plausible. In the non-uniform settings the class of unambiguous nondeterministic log space coincides with the class of nondeterministic log space [10]. It is quite natural to suggest that simplicity test do not belong to LOG if $\text{LOG} \neq \text{NLOG}$.

Remark 4. The unique word accepted by DFA can be easily recovered from special forms of DFA description (say, description of minimal DFA accepting 1-element language). But it does not help to improve the reduction (6) because we are interested in regular realizability problems (the answer depends on a language and should be the same for all automata recognizing the language).

4 Universality of RR problems

In the case of language reductions we are able to prove a weaker version of Theorem 1 using disjunctive reductions.

Theorem 2. *For any non-empty language X there exists a filter L such that*

$$X \leq_m^{\log} \text{RR}(L) \leq_{\text{dtt}}^{\text{FNL}} X. \quad (7)$$

In the proof of Theorem 2 we use the other extreme case for conditions (5). Namely, we choose $Y = X_f \stackrel{\text{def}}{=} \overline{f(\bar{X})}$.

The idea behind the proof is to approximate inversion of a monoreduction as close as possible. The difficulty of inversion stems from the fact mentioned in Remark 4: instances of RR problem are all regular languages and RR problem might be hard for languages that are not 1-element languages constituting the image of a monoreduction. To overcome this difficulty we choose a filter satisfying conditions (5) in such a way that for most regular languages RR problem with the chosen filter is trivial.

The first step toward implementation of this idea is to make RR problem trivial for all infinite languages.

Definition 3. An infinite language is *regularly immune* if it does not contain any infinite regular language.

Suppose that $f(\{0, 1\}^*)$ is contained in some regularly immune language. Then any infinite regular language has a non-empty intersection with $f(\{0, 1\}^*)$. So for any infinite instance of $\text{RR}(X_f)$ the answer is positive.

For finite instance of $\text{RR}(X_f)$ m -reducing algorithm should indicate a word that possibly belongs to X . It seems very hard for arbitrary X .

Therefore the second step is to use disjunctive reductions. In this case reducing algorithm should just produce list of all words accepted by a specific automaton and this task is much easier in general case.

Note that examples of regularly immune languages are numerous and easy to construct. Any such language can be used in reduction outlined above. But the cardinality of a finite language can be exponentially larger than the number of states in DFA recognizing the language. So such a reduction requires an exponential time.

Thus the next step is to choose a regularly immune set D possessing a special property: the cardinality of any regular language in D is polynomially upperbounded by the number of states in DFA recognizing the language. To guarantee a polynomial upper bound it is sufficient to require that D consists of squares only (see Remark 6). Below in Lemma 3 we give a linear bound.

Last but not least, all actions mentioned above should be efficiently implemented. We are going to construct a $\leq_{\text{dtt}}^{\text{FNL}}$ -reduction. So we need FNL implementations.

To realize the plan outlined above we start from specification of D . Let β be a morphism $\beta(0) = 01, \beta(1) = 10$ and $\text{sq}(\cdot)$ be a map

$$\text{sq}: x \mapsto \beta(x)1^20^{|x|^2+3}\beta(x)1^20^{|x|^2+3}. \quad (8)$$

We choose $D = \text{Im}(\text{sq})$, i.e. an image of all binary words under the map sq .

To prove that D is regularly immune one can use Parikh theorem [11, 12]. It says that the lengths of words from a regular (or even CFL) language form a *semilinear set*, i.e. a finite union of arithmetic progressions.

Lemma 2. *D is regularly immune.*

Proof. Lengths of words from D form the set $\{2n^2 + 4n + 10 : n \in \mathbb{N}\}$. It is clear that intersection of this set with any arithmetic progression is finite. \square

To give an upper bound on the cardinality of a regular language contained in D we make a couple of observations.

By definition, D words are squares. Moreover, they are incomparable in the following sense.

Proposition 6. *Let $pq_1, pq_2 \in D$. Then $p \in \beta(\{0, 1\}^*)(\varepsilon \cup \{0, 1\})$.*

Proof. Note that $w = \text{sq}(x)$ can be recovered from the prefix $\beta(x)11$: the first occurrence of 11 starting at even position³⁾ signals that the prefix $\beta(x)$ is completed and x is uniquely determined by this prefix. \square

Proposition 6 will play an important role below. In particular, we will use the fact that common prefix of words from D does not contain 0^3 (easily follows from Proposition 6). Just now we indicate an another simple corollary.

Corollary 2. *No word from D is a prefix of another word from D .*

Remark 5. For an arbitrary language of squares Corollary 2 does not hold: 0101 is a prefix of 010111010111.

Now we are ready to prove a linear bound on the cardinality of a regular language $L(A)$ that sits in D .

Lemma 3. *Let A be DFA such that $L(A) \subset D$. Then $|L(A)| \leq |Q|$, where Q is the state set of A .*

Proof. It is sufficient to consider a minimal DFA B recognizing $L(A)$. The states of B are in one-to-one correspondence with Myhill – Nerode classes. Consider two words $u_1u_2 \neq u_2u_2$ in $L(A)$. We prove that $u_1u_2 \notin L(A)$. It implies that u_1 and u_2 are not equivalent and the number of Myhill – Nerode classes is not less than $|L(A)|$.

Suppose that $u_1u_2 = vv \in L(A)$. Either u_1 is a prefix of v or v is a prefix of u_1 . In both cases we come to a contradiction with Proposition 6: both u_1 and v contain 0^3 as a subword. \square

³⁾We enumerate positions in a word starting with 0.

Note also that lengths of words of a finite regular language $L(A)$ are also linearly upperbounded by the number of states of DFA A .

Proposition 7. *Let A be DFA with the state set Q . If $|L(A)| < \infty$ then for any $w \in L(A)$ we have $|w| < |Q|$.*

This fact can be easily extracted from proofs of pumping lemma for regular languages. An equivalent statement: $L(A)$ is infinite iff there exists an accepting walk on the graph of the automaton A containing a cycle.

Now we construct algorithms used in the proof of Theorem 2.

Note that the binary representation of the length of word x has a size $O(\log |x|)$. So arithmetic operations with numbers of this magnitude can be performed using log space. This fact is widely used below.

Proposition 8. *The membership problem for the language D is in LOG. The map sq and the inverse map sq^{-1} are log space computed.*

Proof. The algorithm that computes sq repeats twice the following procedure: compute $\beta(w)$, where w is the input word, add to the result 1^2 , then add 0s (the required number of zeroes is computed using log space).

The membership test for D can be done in two stages. At first the algorithm checks that an input word is a square uu (easily performed using log space). If so, on the second stage the algorithm finds an occurrence of 11 in the word u starting at even position and splits u in a prefix and suffix removing this subword 11. After that it verifies that the prefix is a code $\beta(x)$ of some x and the suffix is $0^{|x|^2+3}$.

Computing of the inverse map is performed in similar way. \square

In nondeterministic algorithms working with inputs containing a description of DFA A we use procedures ‘guessing a word’ and ‘guessing a square’.

Guessing a word is a repeating guessing of symbols from the alphabet of A . These symbols form a word and in parallel algorithms will test some properties of this word. Tests should use log space and read a word in one-way. Note that DFA description is not shorter than the alphabet size as well as the cardinality of the state set. So, log space is sufficient to store a constant number of symbols and states.

A simple example of a test is a check that a guessing word is accepted by A . This test stores a current state of the automaton and applies the transition function. It has been used above in the proof of Theorem 1.

Guessing a word is also used in algorithmic version of Proposition 7.

Proposition 9. *Infiniteness of a regular language is in NLOG provided a language is represented by DFA recognizing it.*

Proof. The algorithm guesses a state q of an input DFA A . After that it guesses a word $w \in L(A)$ such that the automaton visits q at least twice while reading w . The latter condition is easily verified on log space even if an input should be read in one-way. \square

The next example will be used below.

Proposition 10. *Halves of words from D , i.e. words in the form $\beta(x)1^20^{|x|^2+3}$, can be recognized using log space. Moreover, a recognizing algorithm can read input in one way.*

Proof. The algorithm runs in two stages.

At the first stage it reads pairs of symbols and counts the number of pairs. If the pair 00 is read then the algorithm stops with the negative answer. If the pair 11 is read then the algorithm starts the second stage. In other cases the counter of pairs is increased by 1.

At the second stage the algorithm computes and stores x^2+3 , where x is the counter value in the end of the first stage. Then the algorithm reads symbols and counts the number of symbols. If the symbol 1 is read then the algorithm stops with the negative answer.

After reading the last symbol the algorithm compares the number of 0s read and the stored value $x^2 + 3$. The answer is positive if these values are equal. Otherwise, the answer is negative.

Correctness of the algorithm is clear as well as logarithmic bound on space used. \square

Guessing a square works similarly. It has two input parameters $q_1, q_2 \in Q(A)$. The procedure guesses a word w such that $\delta_A(s, w) = q_1$, $\delta_A(q_1, w) = q_2$ (hence $\delta_A(s, ww) = q_2$). For this purpose it is sufficient to store two states—the current states of reading processes starting at the state s and at the state q_1 respectively. In parallel tests for the guessed word might be launched. They also should use log space and read a word in one-way.

Guessing a square is used in the following algorithm that checks triviality of RR problem for a finite language. This check will be applied for RR problems with a filter containing \bar{D} . Thus for instances A such that $L(A) \setminus D \neq \emptyset$ the answer of RR problem is positive.

Lemma 4. *Testing conjunction $|L(A)| < \infty$ and $L(A) \subset D$ is in NLOG.*

Proof. Testing finiteness of a regular language is in NLOG (Proposition 9). So the algorithm in question performs this check at the first stage. If $L(A)$ is infinite the algorithm outputs the answer and finishes. So in the sequel we assume that $L(A)$ is finite.

Note that $L(A) \setminus D \neq \emptyset$ iff

- either there exists $w \in L(A)$ of odd length;
- either there exists $w \in L(A)$ of even length that is not a square;
- or all words in $L(A)$ are squares and there exists $ww \in L(A) \setminus D$.

The algorithm guesses among these three cases and tests a chosen property.

Note that by Proposition 7 lengths of words from $L(A)$ are not greater than input size (i.e. DFA description). So counting up to a word length can be done using log space.

In the first case the algorithm guesses a word $w \in L(A)$ of odd length (and check the parity of the length).

In the second case it guesses position i , the length $2k$ and word w of the length $2k$ from $L(A)$ such that i th and $(k+i)$ th bits of w are different (and uses counters to find these bits).

In the last case it guesses a square $ww \in L(A) \setminus D$ and checks that $ww \notin D$ using Proposition 10. Parameters are states q, t such that $\delta_A(s, w) = q$ and $\delta_A(q, w) = t$. The algorithm guesses them before guessing a square.

Correctness of the algorithm follows from the above observations. \square

Our next goal is an algorithm that extracts words accepted by an automaton recognizing a finite language and outputs their images under inverse map sq^{-1} . We assume here that the regular language in question sits in D .

Let A be DFA such that $L(A) \subset D$, Q be the state set, s be the initial state and Q_a be the set of accepting states of A . We denote by δ_A the transition function $\delta: Q \times A^* \rightarrow Q$ of A extended to the set of words in the input alphabet of A in natural way.

Suppose that all words in $L(A)$ are squares. Define a map

$$\mu: L(A) \rightarrow Q \tag{9}$$

by the rule: if $w = uu \in L(A)$ then $\mu(w) = \delta_A(s, u)$.

Map (9) is not injective. But its preimage is bounded.

Proposition 11. *If $L(A)$ is a square language then for any accepting state t , integer k and a state q there exists at most one word $uu \in \mu^{-1}(q)$ such that $|u| = k$ and $\delta_A(s, uu) = t$.*

Proof. By contradiction. If two different squares uu, vv satisfy these conditions then $uv \in L(A)$ but is not a square. \square

Remark 6. Proposition 11 implies a polynomial bound of the cardinality of $L(A)$ consisting of squares.

From propositions 7 and 11 we get a bound

$$|L(A)| = \sum_{q \in Q} |\mu^{-1}(q)| \leq \frac{|Q|}{2} \cdot |Q| \cdot |Q| = |Q|^3/2.$$

Lemma 5. *There exists an FNL algorithm that outputs the list of words of $\text{sq}^{-1}(L(A))$ provided $L(A) \subset D$.*

Proof. It follows from Proposition 11 that each word $uu \in L(A)$ is uniquely determined by states $q \in Q$, $t \in Q_a$ and integer k such that $\mu(uu) = q$, $|u| = k$, $\delta_A(s, uu) = t$. Proposition 7 guarantees that $k < |Q|$.

The algorithm in question tries all possible values q, t, k . For each triple it uses an NLOG oracle to check an existence of a word $uu \in D$ determined by the triple.

An NLOG algorithm for this check guesses a square with parameters q, t . In parallel it verifies that the length of a guessed word u is k and $uu \in D$. For the former test it uses a counter of guessed symbols. The latter is possible due to Proposition 10.

If $uu \in D$ for a triple q, t, k does exist then the algorithm decodes x such that $u = \beta(x)1^{2|x|^2+3}$ and outputs x .

It can be done by guessing a word u combined with two simulations of reading the word u by the automaton A . One simulation starts from the initial state s and the another starts from q . These two simulations need to store two current states q_1, q_2 respectively.

Guessing a symbol is replaced in this procedure by two trials. For each possible variant α of the next symbol (there are two of them) the modified algorithm simulates reading the pair $\alpha\bar{\alpha}$ and for new values q'_1 and q'_2 asks an NLOG oracle about existence of a word w such that $\delta_A(q'_1, w) = q, \delta_A(q'_2, w) = t$. The corresponding NLOG algorithm is a modification of guessing a square. Due to $\text{NLOG} = \text{co-NLOG}$ an negative answer can be also verified on nondeterministic log space.

The word uu is unique for the triple q, t, k . So the oracle answers positively for exactly one value of α . This value is the next symbol of x and the algorithm outputs it. \square

Tying up loose ends we get the proof of the main result.

Proof of Theorem 2. Let us prove that

$$X \leq_m^{\log} \text{RR}(X_{\text{sq}}) \leq_{\text{dtt}}^{\text{FNL}} X. \quad (10)$$

The first reduction does exist due to (4) and Proposition 8.

Now we construct the second reduction. Let A be an instance of $\text{RR}(X_{\text{sq}})$. The reducing algorithm checks infiniteness of $L(A)$ and $L(A) \setminus D \neq \emptyset$ using Lemma 4. If R is infinite or it contains a word from \bar{D} then the reducing algorithm forms a query list of length 1 containing a fixed element $x_0 \in X$ and output the list.

Otherwise the algorithm outputs the list of all words from $\text{sq}^{-1}(L(A))$ applying the algorithm from Lemma 5.

To prove correctness of the above reduction note that D is regularly immune. So any infinite regular language has a common word with X_{sq} . If $L(A)$ is finite and contains a word from \bar{D} , then it has nonempty intersection with X_{sq} . Finally, if a finite language $L(A)$ is contained in D then $R \cap X_{\text{sq}} \neq \emptyset$ iff the output list contains a word from X . \square

Taking into account Statement 5 we get the following corollary.

Corollary 3. *Each class Σ_k^p, Π_k^p of the polynomial hierarchy contains an RR problem that complete for a class under \leq_m^{FNL} -reductions (and under polynomial reductions).*

5 Universality of generalized nondeterministic models

RR problems are closely related to models of generalized nondeterminism (GNA) introduced in [3]. GNA classes are parametrized by languages of infinite words (certificates). It was shown in [1] that each GNA class contains RR problem complete for the class under \leq_m^{\log} -reductions. A filter of RR problem is formed by prefixes of certificates for the GNA class.

Note that filters in the proof of Theorem 2 are not prefix closed. So they do not correspond any GNA class. To prove universality of GNA classes we need a more sophisticated construction.

What RR problems correspond to GNA classes? To be prefix closed is a necessary condition only. The second condition is the following: each filter word is a proper prefix of a filter word. These two conditions guarantee that the filter is the prefix set for some set of certificates.

To satisfy the second condition a very simple modification of a filter is needed.

Proposition 12. *If $L \subseteq \Sigma^*$, $\# \notin \Sigma$ then*

$$\text{RR}(L) \leq_m^{\log} \text{RR}(L\#^*) \leq_m^{\log} \text{RR}(L).$$

Proof. The language L is a regular restriction of $L\#^*$. We get the first reduction from this observation and [1, Lemma 4].

Now we construct the second reduction. Let A be an instance of the problem $\text{RR}(L\#^*)$. To output an instance B of $\text{RR}(L)$ the reducing algorithm changes the set of accepting states of DFA A only. A state q is accepting for the automaton B iff $\delta_A(q, \#^k)$ is an accepting state of A for some k .

To check correctness of this reduction take a word in the form $w\#^k$, $w \in L \cap L(A)$. Then B accepts w by construction. In the other direction: if B accepts $w \in L$ then after reading w the automaton A moves to a state such that an accepting state is reachable by reading a sequence of $\#$.

Note that $\text{RR}(\#^*) \in \text{LOG}$ [1, Corollary 1]. Using this observation a log space algorithm for the second reduction can be easily constructed. \square

Thus for any RR problem with a prefix closed filter there exists an equivalent RR problem with a filter that coincides with the prefix set of a language of infinite words.

So universality of GNA classes follows from the following generalization of Theorem 2.

Theorem 3. *For any nonempty language X there exists a prefix closed filter P such that*

$$X \leq_m^{\log} \text{RR}(P) \leq_{\text{dtl}}^{\text{FNL}} X. \quad (11)$$

To prove Theorem 3 we follow the same plan as for Theorem 2.

Again we use monoreductions. We choose the filter P in the form X'_f for some map f and some set X' which is \leq_m^{\log} -equivalent to X . Actually we will use $X' = 0X \cup \{\varepsilon\}$.

To make the filter P prefix closed we choose an appropriate map f . Namely, we require that f maps words in the form $0\{0, 1\}^*$ to a prefix code D (i.e. a set of words such that no word in the set is a prefix of another word in the set). Also we require that f maps all nonempty words to the set $D^\dagger = \{w : w = uv, u \in D\}$ of all possible extensions of D words. Then $f(\overline{X'})$ is suffix closed while $P = X'_f = \overline{f(\overline{X'})}$ is prefix closed.

By construction P contains \bar{D}^\dagger . So for an instance A of the RR problem $\text{RR}(P)$ the answer is positive if $L(A) \setminus D^\dagger \neq \emptyset$. The latter condition plays now a role of infiniteness condition in the proof of Theorem 2.

To keep arguments from the previous section a check of $L(A) \setminus D^\uparrow \neq \emptyset$ should belong to NLOG and generation of the list containing all prefixes of words from $L(A) \cap D$ should be done by a FNL algorithm provided $L(A) \subset D^\uparrow$.

Now we elaborate details of this plan.

We use the set D from the previous section as a prefix code (see Corollary 2).

The set D^\uparrow is not regularly immune. But in the crucial case $L(A) \subset D^\uparrow$ (see the above plan) the D -prefix set

$$R_D = \{w : w \in D \text{ and } wv \in L(A)\}$$

is finite. Below we modify algorithms from Section 1 to deal with the sets of this form.

Now we upperbound the cardinality of R_D and lengths of words in R_D .

Proposition 13. *If $L(A) \subset D^\uparrow$, where A is DFA with the state set Q , $|Q| = n$, and $w \in R_D$ then*

$$|w| \leq 2(n-3) + 4\sqrt{n-3} + 10.$$

Proof. By definitions a word $w \in D$ has a form

$$\beta(x)1^20^{\ell^2+3}\beta(x)1^20^{\ell^2+3},$$

where $\ell = |x|$ and $wz \in L(A)$ for some z .

If $\ell^2 + 3 \geq n$ then while reading the word 0^{ℓ^2+3} the automaton visits some state at least two times. Thus for some integer $k > 0$ and all integers $i \geq 0$ we have

$$w_i = \beta(x)1^20^{\ell^2+3+ik}\beta(x)1^20^{\ell^2+3}z \in L(A).$$

Note that for $i > 0$ a word w_i does not have a D prefix in contradiction with the hypothesis $L(A) \subset D^\uparrow$. Indeed such a prefix must have a prefix $\beta(x)1^2$ and due to Proposition 6 must coincide with the word w . But w is not a prefix of w_i for $i > 0$.

Thus $\ell^2 + 3 < n$. The length of w equals $2\ell^2 + 4\ell + 10$. It implies the required inequality. \square

To upperbound the cardinality of R_D we modify Proposition 11.

Proposition 14. *Let $L(A) \subset D^\uparrow$. For any coaccessible⁴⁾ state t , integer k and a state m there exists at most one word $uu \in D$ such that $|u| = k$ and $\delta_A(s, u) = m$, $\delta_A(m, u) = t$.*

Proof. Suppose there are two words uu and vv satisfying these conditions. Then $uvw \in L(A)$ for some w . From Proposition 6 we conclude that a word uvw does not contain D prefixes in contradiction with the hypothesis $L(A) \subset D^\uparrow$. \square

Corollary 4. *If $L(A) \subset D^\uparrow$ then $|R_D| \leq 2|Q|^3(1 + o(|Q|))$.*

Correctness of the algorithm from Lemma 5 is based on Proposition 11. Replacing it by Proposition 14 we get the following lemma.

⁴⁾A coaccessible state is a state from which it exists a path to an accepting state.

Lemma 6. *There exists a map $f \in \text{FNL}$ that generates the element list of $\text{sq}^{-1}(R_D)$ provided $L(A) \subset D^\uparrow$.*

Proof. The algorithm mimics the algorithm from Lemma 5 but for guessed squares it tests a membership to prefixes of $L(A)$ (instead of a membership to $L(A)$ itself). The prefix set of $L(A)$ is recognized by DFA A_p having the same state set and the transition table as DFA A . Accepting states of A_p are coaccessible states of A .

It is possible to check coaccessibility on nondeterministic log space. So a FNL algorithm is able to apply this test.

Correctness of the algorithm follows from Proposition 14. \square

Lemma 7. *Testing $L(A) \subset D^\uparrow$ is in NLOG.*

Proof. The statement is equivalent to $\text{RR}(\overline{D^\uparrow}) \in \text{NLOG}$ due to equality $\text{NLOG} = \text{co-NLOG}$.

The algorithm is similar to the algorithm from Lemma 4.

Let DFA A with the state set Q be an instance of $\text{RR}(\overline{D^\uparrow})$. The algorithm guesses a word from $\overline{D^\uparrow} \cap L(A)$ and verifies this condition. It uses log space and reads an input in one way.

To detect a word from $\overline{D^\uparrow} \cap L(A)$ the algorithm guesses a prefix of a word $w \in L(A)$ certifying that $w \notin D^\uparrow$.

At first a prefix in the form

$$\beta(x)(\varepsilon \cup \{0, 1\}), \quad |x| = 5|Q|, \quad (12)$$

guarantees that $L(A) \setminus D^\uparrow \neq \emptyset$ due to Proposition 13.

Note that each language from the following list

$$\beta(x)00, \quad (13)$$

$$\beta(x)110^i1, \quad i < |x|^2 + 3, \quad (14)$$

$$\beta(x)110^i, \quad i > |x|^2 + 3, \quad (15)$$

$$\beta(x)110^{|x|^2+3}\beta(y)00, \quad (16)$$

$$\beta(x)110^{|x|^2+3}\beta(y), \quad |x| < |y|, \quad (17)$$

$$\beta(x)110^{|x|^2+3}\beta(y)11, \quad |x| > |y|, \quad (18)$$

$$\beta(x)110^{|x|^2+3}\beta(y)110^i1, \quad i < |y|^2 + 3, \quad |x| = |y|, \quad (19)$$

$$\beta(x)110^{|x|^2+3}\beta(y)110^i, \quad i > |y|^2 + 3, \quad |x| = |y|. \quad (20)$$

do not contain neither words having a prefix from D nor prefixes of words from D .

Also, each condition (12–20) is recognized by a log space algorithm reading an input in one way provided the length of a word is less then $10|Q(A)| + 1$.

With the same restrictions it is possible to verify that the language $L(A)$ contains a word w such that w is a prefix of a word from a language presented in the list (12–20). More exactly, for (20) we exclude prefixes in the form

$$\beta(x)110^{|x|^2+3}\beta(y)110^{|x|^2+3}, \quad |x| = |y|. \quad (21)$$

If $L(A)$ do not contain any word satisfying the above requirements then each word from $L(A)$ has a prefix in the form (21). In this case the algorithm should indicate a prefix with $x \neq y$. If such a prefix does not exist we have $L(A) \subset D^\uparrow$.

To indicate a prefix in the form (21) with $x \neq y$ the algorithm guesses two states q, t such that t is coaccessible (and checks coaccessibility by consulting with NLOG oracle).

The algorithm also guesses states

$$q_1 = \delta_A(s, \beta(p)), \quad q_2 = \delta_A(q, \beta(p)), \quad (22)$$

where p is the greatest common prefix of x and y . Then it simulates reading $\beta(p)$ starting from states s and q keeping the number of guessed symbols in a counter. It continues by reading 01 in one case and by reading 10 in the another (to guarantee that $x \neq y$). After that the algorithm guesses two words separately verifying for both of them the format (21) and counting their lengths. When guessing is finished the algorithm checks that the above conditions (22) and compares the lengths of words. \square

Now we present a map sq^\uparrow to construct the second reduction in (11):

$$\begin{aligned} \text{sq}^\uparrow: \varepsilon &\mapsto \varepsilon, \\ \text{sq}^\uparrow: 0x &\mapsto \text{sq}(x), \\ \text{sq}^\uparrow: 1x &\mapsto \text{sq}(\pi_1(x))\pi_2(x), \end{aligned} \quad (23)$$

where $\pi(x) = (\pi_1(x), \pi_2(x))$ is a bijection $\{0, 1\}^*$ onto $\{0, 1\}^* \times (\{0, 1\}^* \setminus \{\varepsilon\})$ such that π, π^{-1} are log space computable.

It immediately follows from definition that sq^\uparrow is injective and $\text{Im}(\text{sq}^\uparrow) = D^\uparrow \cup \{\varepsilon\}$. It easy to see that sq^\uparrow and $\text{sq}^{\uparrow^{-1}}$ are log space computable.

Note that a bijection π in (23) can be easily constructed from a map

$$\varphi: (x, \alpha y) \mapsto \beta(x)\alpha\alpha y, \quad (24)$$

which sends $\{0, 1\}^* \times (\{0, 1\}^* \setminus \{\varepsilon\})$ to a set S consisting of words of the length greater than 3 and words 00, 11. Namely, $\pi(x) = \varphi^{-1}(\iota(x))$, where ι is an injective map to S such that both ι and ι^{-1} are log space computable.

For the sake of completeness we presented a construction of such a map.

Lemma 8. *Let S be a cofinite set of binary words, i.e. $|\bar{S}| < \infty$. Then there exists an injective map f_S such that domain of f_S is the set of all words, $\text{Im}(f_S) = S$ and f_S, f_S^{-1} are log space computable.*

Proof. Choose a set of binary words M such that all words in M have the same length n , $|M| = |\bar{S}|$ and

$$n > \max_{x \in \bar{S}} |x|.$$

Then choose a bijection $\varkappa: \bar{S} \rightarrow M$. Now a map f_S in question can be defined as follows

$$\begin{aligned} f_S: x &\mapsto 1\varkappa(x), & \text{for } x \in \bar{S}, \\ f_S: 0^i 1m &\mapsto 0^{i+1} 1m, & \text{for } q \in M, \\ f_S: x &\mapsto x & \text{otherwise.} \end{aligned}$$

It is obvious that f_S satisfies all requirements. \square

Proof of Theorem 3. It is observed above that the language X is equivalent to the language $X' = 0X \cup \{\varepsilon\}$ under \leq_m^{\log} -reductions. From definition (23) the map sq^\uparrow sends nonempty words from X' to the set D while the images of all nonempty words coincide with D^\uparrow . It means that $\text{sq}^\uparrow(\overline{X'})$ is suffix closed, i.e. $w \in \text{sq}^\uparrow(\overline{X'})$ implies $wv \in \text{sq}^\uparrow(\overline{X'})$ for all v , and $X'_{\text{sq}^\uparrow} = \text{sq}^\uparrow(\overline{X'})$ is prefix closed.

Therefore it is sufficient to show that

$$\text{RR}(X'_{\text{sq}^\uparrow}) \leq_{\text{dtt}}^{\text{FNL}} X'.$$

The reduction is constructed using Lemmata 6 and 7 in a way similar to the proof of Theorem 2.

Using Lemma 7 the reducing algorithm checks $L(A) \subset D^\uparrow$ for an automaton A which is an instance of the problem $\text{RR}(X'_{\text{sq}^\uparrow})$. If $L(A) \not\subset D^\uparrow$ then the answer for this instance is positive and the reducing algorithm outputs a trivial query list containing the fixed word from X'_{sq^\uparrow} . Otherwise the reducing algorithm invokes the procedure from Lemma 6 to generate the element list of $\text{sq}^{-1}(R_D)$. It follows from construction of the set X'_{sq^\uparrow} that $L(A) \cap X'_{\text{sq}^\uparrow} \neq \emptyset$ iff $x_j \in X'$ for some j . \square

Taking into account normality of classes in the polynomial hierarchy we get the following corollary.

Corollary 5. *For each k the classes Σ_k^P, Π_k^P contain RR problems with a prefix-closed filter that are complete for a class under polynomial reductions.*

References

- [1] Vyalyi M.N. On regular realizability problems. Problems of Information Transmission. Vol. 47, issue 4, 2011. P. 342–352.
- [2] Anderson T., Loftus J., Rampersad N., Santean N., Shallit J. Detecting palindromes, patterns and borders in regular languages Information and Computation. Vol. 207, 2009. P. 1096–1118.
- [3] Vyalyi, M.N., On Nondeterminism Models for Two-Way Automata, in Proc. VIII Int. Conf. on Discrete Models in Control System Theory, Moscow, 2009, Moscow: MAKS Press, 2009, pp. 54–60.
- [4] Arora S., Barak B. Computational complexity: a modern approach. Cambridge, UK: Cambridge Univ. Press, 2009.
- [5] Sipser M. Introduction to the theory of computation. 3rd edition. Independence, KY, USA: Course Technology, 2012. — 480 .
- [6] Ladner R. E., Lynch N. A., Relativization of questions about log space computability. Mathematical Systems Theory. Vol. 10, no. 1, 1976. P. 19–32.

- [7] *Álvarez C., Balcázar J. L., Jenner B.* Adaptive logspace reducibility and parallel time. *Mathematical Systems Theory*. Vol. 28, no. 2, 1995. P. 117–140.
- [8] *Allender E., Ogihara M.* Relationships among PL, #L, and the determinant. *Informatique Théorique et Applications*. Vol. 30, 1996. P. 1–21.
- [9] *Immerman N.* Nondeterministic space is closed under complement. *SIAM Journal on Computing*. Vol. 17, no. 5, 1988. P. 935–938.
- [10] *Reinhardt K., Allender E.* Making nondeterminism unambiguous. *SIAM Journal on Computing*. Vol. 29, no. 4, 2000. P. 1118–1131.
- [11] *Kozen D.* Automata and Computability. New York: Springer-Verlag, 1997.
- [12] *Parikh R. J.* On context-free languages. *Journal of the ACM*. Vol. 13, no. 4, 1966. P. 570–581.
- [13] *Hopcroft D., Motwani R., Ullman J. D.* Introduction to Automata Theory, Languages, and Computation. Addison Wesley, 2000.